



IOS Win32 Development Package User's Manual

**ACROMAG INCORPORATED
30765 South Wixom Road
P.O. BOX 437
Wixom, MI 48393-7037 U.S.A.**

**Tel: (248) 295-0310
Fax: (248) 624-9234**

**Copyright 2009 - 2010, Acromag, Inc., Printed in the USA.
Data and specifications are subject to change without notice.**

9500-390B

The information in this document is subject to change without notice. Acromag, Inc., makes no warranty of any kind with regard to this material and accompanying software, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Further, Acromag, Inc., assumes no responsibility for any errors that may appear in this document and accompanying software and makes no commitment to update, or keep current, the information contained in this document. No part of this document may be copied or reproduced in any form, without the prior written consent of Acromag, Inc.

Copyright 2009 - 2010, Acromag, Inc.

All trademarks are the property of their respective owners.

Contents

Contents	3
Introduction	4
Hardware Support	5
Windows Embedded Standard 2009 Image	6
Initial Setup.....	6
I/O Server Installed Software.....	7
Installing Third-Party Applications	8
IOS Win32 Driver Software (IOSSW-API-WIN)	9
Software Installation	9
Development System Installed Software.....	9
Enumeration Utility	9
Language Support.....	9
Software Overview	10
Function Format.....	10
Status Codes.....	11
Sequence of Operations.....	13
Interrupts	14
Callback Functions	15
PCI Event ActiveX control	16
Synchronization	17
Base Address Pointers	17
Building Windows Applications	18
C/C++	19
Microsoft Visual C++ 6	19
Microsoft Visual C++ .NET	19
Visual Basic .NET	20
LabVIEW.....	22
Deploying Applications to the I/O Server	28
Debugging Applications on the I/O Server	28
Windows Embedded Standard Topics.....	29
Target Analyzer	29
I/O Carrier Driver	29
System On Module Drivers	31
I/O Carrier and Module DLLs	31
DLL Location Notes.....	31
PCI Event ActiveX Control	32

Introduction

The IOS Win32 Development Package includes two compact discs.

IOS Hardware Manuals (IOS-MANUALS)

This disc includes instruction manuals for the I/O Server and each IOS module in PDF format.

IOS Win32 Driver Software (IOSSW-API-WIN)

This software is used to develop custom applications that interface with the IOS Carrier and IOS Modules. It should be installed on the Development System (rather than the I/O Server). After creating an application on the Development System, it is deployed to the I/O Server.

Applications interface with the IOS Carrier and IOS Modules through exported DLL functions. DLL functions use the Windows `_stdcall` calling convention and can be accessed from a number of programming languages. In addition to the DLLs, the software includes an ActiveX control for implementing interrupt notifications in programming environments that do not support the use of callback functions. Several example C, Visual Basic .NET and LabVIEW applications are provided with source code.

This user's manual begins with an overview of the **Windows Embedded Standard** images that are included with the **IOS-7200-WIN** and **IOS-7400-WIN** I/O Server products. The remainder of the manual covers general information on IOS Win32 Driver Software installation, programming concepts, application development and deployment issues. The following documents are also installed with the IOS Win32 Driver Software:

- Function Reference document for the IOS Carrier DLL
- Function Reference documents for each IOS module DLL

After reviewing this user's manual, readers will next want to consult the Function Reference documents specific to their hardware.

Hardware Support

Supported Acromag I/O modules are shown in Table1.

Table 1: Acromag I/O Server Modules

Model	Description	Interrupts
IOS-220-X	Analog output, 12-bit D/A, (-8, -16) 8 or 16 channels	No
IOS-231-X	Analog output, 16-bit D/A, (-8, -16) 8 or 16 channels	No
IOS-320	Analog input, 12-bit A/D, 20D / 40SE channels	No
IOS-330	Analog input, 16-bit A/D, 16D / 32SE channels	Yes
IOS-341	Analog input, 14-bit A/D simultaneous sample and hold, 16DE	Yes
IOS-408	Digital input/output, 32 high-voltage bi-directional I/O	Yes
IOS-409	Digital input/output, 32 differential bi-directional I/O	Yes
IOS-440-2	Digital input, 32 port-isolated channels	Yes
IOS-445	Digital output, 32 port-isolated channels (SSRs)	No
IOS-470	Digital input/output, 32 TTL bi-directional I/O	Yes
IOS-482	Counter/timer, 16 Bit, 10 TTL channels	Yes
IOS-483	Counter/timer, 16 Bit, 2 RS422 and 5 TTL channels	Yes
IOS-484	Counter/timer, 16 Bit, 5 RS422 channels	Yes
IOS-520	Serial communication, 8 EIA/TIA-232E ports	Yes
IOS-521	Serial communication, 8 EIA/TIA-422/485 ports	Yes
IOS-EP201	Re-configurable FPGA, digital I/O, 48 TTL I/O lines	Yes
IOS-EP202	Re-configurable FPGA, digital I/O, 24 differential RS485 I/O lines	Yes
IOS-EP203	Re-configurable FPGA, digital I/O, 24 TTL and 12 RS485 I/O lines	Yes
IOS-EP204	Re-configurable FPGA, digital I/O, 24 LVDS I/O lines	Yes
IOS-560	Two Controller Area Network (CAN) 2.0B channels	Yes
IOS-571/572	1 and 2 channel MIL-STD-1553 interface modules	Yes

Windows Embedded Standard 2009 Image

A licensed Windows Embedded Standard 2009 image is provided on Compact Flash with the **IOS-7200-WIN** and **IOS-7400-WIN** I/O Server products. The images are configured for the AMD Geode and Intel Atom processor systems respectively.

Console demonstration programs and DLLs for all IOS modules are pre-installed. This allows users to quickly test installed modules and simplifies the deployment of custom applications.

Initial Setup

1. Connect the monitor, keyboard and mouse to the I/O Server. The carrier card should be installed but IOS modules need not be present.
2. Insert the Compact Flash card with the Windows Embedded Standard image into the slot in the back of the unit.
3. Apply power to the system.
 - a. The systems boots and a "Resealing in progress" dialog is displayed for several minutes. During this process the system receives a unique security ID (SID) and Computer Name.
 - b. The system finishes booting to the desktop and a readme file is displayed. Depending on attached peripherals, the user may or may not be prompted to restart the system in order to finish the installation of new devices.
4. Either click the Yes button to restart the system or reboot the system manually. **This reboot is necessary to start the IIS service.** After the system reboots it is ready to use.

Note:

- **After completing the Initial Setup, Acromag strongly recommends you use disk imaging software to make a backup of the Compact Flash card.**

I/O Server Installed Software

DLLs and demonstration programs for all IOS modules are pre-installed on the system. This allows users to quickly test installed modules and simplifies the deployment of custom applications. Files are installed under C:\Program Files\Acromag\IO Server

Subdirectory	
<none>	Readme file – also available from Program Files menu
carrier_driver	The IOS carrier kernel driver and INF file
config_files	Example VHDL object code for reconfigurable IOS modules
data_files	Miscellaneous files. See files.txt .
demos	Executable console demonstration programs for all IOS modules
libraries	DLLs and ActiveX control files
utility	IOS module enumeration utility – also available from Program Files menu

Notes:

- The I/O Server Readme file is displayed automatically during each system start. To prevent this from happening, delete the shortcut to the file from the Startup folder:

C:\Documents and Settings\All Users\Start Menu\Programs\Startup

The Readme file will still be available from the Program Files menu.
- The path to the “libraries” directory has been appended to the PATH environment variable. This allows the operating system to find and load the DLLs without placing them in Windows\system32.
- The above files are also included with **IOS Win32 Driver Software (IOSSW-API-WIN)** and are placed on the Development system during installation of that software. The files are discussed in more detail in the **IOS Win32 Driver Software** section of this document.

In addition to the Acromag software listed above, several useful utilities are available from the **All Programs > Advantech** menu:

- **Advantech Booting Manager** – This utility is used to the configure auto logon and boot logo settings.
- **Advantech EWF Manager** – The Enhanced Write Filter Manager can be used to write-protect the run-time image by redirecting all write requests to a volume to RAM. The Manager can also be used to activate/deactivate the HORM (Hibernate Once Resume Many) feature.
- **Advantech FBWF Manager** – The File-Based Write Filter Manager is also used to write-protect the run-time image by redirecting write requests to RAM. However, FBWF operates at the file level rather than the sector level.

When write protecting a volume, one would use either EFW or FBWF, but not both. Generally, FBWF is the preferred choice. Advantech's user manuals for the EWF and FBWF Manager are included with **IOS Win32 Driver Software (IOSSW-API-WIN)** and are placed on the Development system during installation of that software. More detailed information about EWF, HORM and FBWF is readily available from online sources such as the Microsoft Developers Network.

Installing Third-Party Applications

Windows Embedded Standard 2009 is a componentized version of Windows XP Professional SP3. Standard allows the system designer to include only the components necessary, which in turn reduces the operating system footprint.

Embedded Standard 2009 and XP Professional SP3 are binary compatible so any application that runs on XP Professional can run on Standard, but only if all the components needed by the application are present.

Consequently, a third-party application may or may not install and run successfully on the I/O Server's Windows Embedded Standard 2009 image.

Since an application that does not install correctly may be difficult to remove cleanly, Acromag strongly recommends you use disk imaging software to make a backup of the Compact Flash card prior to attempting to install any third-party applications.

If a third-party application does not install or run properly, there are number of free utilities available to help determine the files and registry entries it requires. These utilities are used to gather information about the successful installation and execution of the application on a system running XP Professional.

InCtrl5 (www.pcmag.com)

This utility generates a report which lists changes made to a system by an application's installation program.

Dependency Walker (www.dependencywalker.com)

Dependency Walker is used to identify the modules (exe, dll, ocx, sys, etc.) called by an executable.

Process Explorer (technet.microsoft.com)

Process Explorer displays file use and registry access information for an application.

If the I/O Server's Windows Embedded Standard 2009 image is only, for example, missing a single DLL required by the third-party application, it may be possible to "patch" the system with a DLL copied from an XP SP3 system. Note, however, that this isn't guaranteed to work as the copied file may not be the correct version.

If there are multiple components or registry entries missing that the application relies on, patching the I/O Server may not be a realistic solution. In this situation the first option would be to find an acceptable substitute for the problematic application. If that's not possible, another option would be to develop your own Windows Embedded image that includes the necessary infrastructure for the application using Microsoft Windows Embedded Studio. See the **Windows Embedded Standard Topics** section for additional information.

IOS Win32 Driver Software (IOSSW-API-WIN)

Software Installation

To install the IOS Win32 Driver software, insert the IOS Win32 Driver Software disk into the CD drive on the development system (not the I/O Server) and run **Setup.exe**. Note that administrative rights are required to perform the installation.

DEVELOPMENT SYSTEM INSTALLED SOFTWARE

The default installation directory is C:\Acromag\IOSSW_API_WIN.

Subdirectory	
c_examples	Microsoft Visual C++ examples with source code. Prebuilt executables for each module are located under msdev\Release.
c_include	Header files
c_lib	COFF format import libraries
config_files	Example VHDL object code for reconfigurable IOS modules
data_files	Miscellaneous files. See files.txt .
docs	User's manuals and DLL reference documents
labview examples	LabVIEW 6i and 7 examples with source code
redist\carrier_driver	Carrier kernel driver and INF file
redist\DLLs	IOS carrier and module DLLs, Microsoft® C Runtime and MFC DLLs
redist\system_on_module	System drivers for SOM boards
utility	IOS module enumeration utility (IEnum.exe)
vb_examples	Visual Basic .NET examples with source code
vb_include	Visual Basic function prototype modules

In addition, the PCI Event ActiveX control is installed to

C:\Program Files\Common Files\Acromag\PCIEvent

This control is shared by IOSSW-API-WIN, IPSW-API-WIN and PCISW-API-WIN.

ENUMERATION UTILITY

The Enumeration utility (IEnum.exe) is a command line utility that may be run to display basic information about the I/O Module Carrier. This information includes the carrier number, bus number, device number, vendor ID, device ID, PCIBAR0 base address, Irq, and the names of all installed IOS modules. In addition, the utility indicates if the carrier supports IOS memory space and 32MHz operation. The kernel driver and carrier DLL (APC86xx.dll) must be installed to use this utility.

Language Support

IOS Win32 Driver Software has been tested with the following development environments:

- Visual C++ 6.0, .NET 2003, 2005, 2008 and 2010
- Visual Basic .NET 2003, 2005 and 2008
- National Instruments LabVIEW 6i and 7

Software Overview

The software includes a single Windows 32 DLL for carrier access and DLLs for each IOS module. In most cases the name of the DLL matches the name of the IOS module. There are a few exceptions, however, where groups of similar IOS modules are supported by a single DLL. These include:

IOS Modules	Shared DLL
IOS-EP201, -EP202, -EP203, -EP204	I1K100.dll
IOS-482, -483 and -484	I482.dll
IOS-571 and IOS-572	I571.dll (and emacr_32.dll)

The DLLs provide the Application Programming Interface (API) used to access the hardware. Each DLL is written in C and contains functions using the `_stdcall` calling convention. A DLL is loaded and linked at runtime when its functions are called by an executable application. Multiple applications can access the functions of a single copy of a DLL in memory.

In addition to the DLLs, the software also includes an ActiveX control that may be used to implement interrupt notifications in programming environments that do not support the use of callback functions.

Note:

Each Acromag IOS module has a corresponding Acromag IndustryPack® module that shares the same memory map (e.g. IOS-482 and IP482). The IOS Carrier also shares a memory map with Acromag's PCI and CompactPCI IndustryPack carriers. As a result, **IOS Win32 Driver Software** and Acromag's IndustryPack Windows software, **IP Win32 Driver Software**, include many of the same DLLs. The usage of the software is essentially the same for IOS and IP applications with the exception that an IndustryPack application may need to interface with more than one carrier. An application written for Acromag IndustryPacks on a single carrier should not need any modifications to run on an I/O Server equipped with the corresponding IOS modules.

FUNCTION FORMAT

All carrier DLL functions have the following form:

```
status = A86_FunctionName(arg1, arg2, ... argn)
```

The format of IOS module DLL functions is similar:

```
status = IXXX_FunctionName(arg1, arg2, ... argn)
```

The "IXXX" portion of the function name indicates the IOS module the function is used with (e.g. I470).

Every function returns a 16-bit status value. This value is set to 0 when a function completes successfully or to a negative error code if a problem occurred. The following **Status Codes** section describes the values that may be returned from the DLL functions.

For most functions, *arg1* is an integer "handle" used to reference the carrier or a specific IOS module. (See the **Sequence of Operations** section below.)

Status Codes

The table below summarizes the 16-bit status codes that may be returned from the DLL functions and ActiveX control methods. Please note the return code of any failing functions if you should need to contact Acromag for technical support.

Value	Mnemonic	Description
0	OK	Operation Successful
-1	E_INVALID_MODHNDL	Returned if no IOS module is associated with the specified handle. Applies to most IOS DLL functions.
-2	E_CARD_IN_USE	Returned by <i>A86_Open</i> if card is already open. This can occur if the carrier is in use by another application.
-3	E_NEWDEV	Returned by <i>A86_Open</i> if an error occurred creating a software instance of the device
-4	E_CONNECT	Returned by <i>A86_Open</i> if an error occurred connecting to the carrier. This will occur if the specified card number is invalid or if the kernel mode drivers are not properly installed or configured.
-5	E_MAPMEM	Returned by <i>A86_Open</i> or if an error occurred mapping the devices physical memory into the virtual address space.
-6	E_THREAD	Returned by <i>A86_Open</i> if an error occurred while creating the interrupt service routine thread.
-7	E_ISR_ENABLE	Returned by <i>A86_Open</i> if an error occurred while enabling interrupt support for the carrier.
-8	E_OUTOFHANDLES	Returned by <i>A86_Open</i> or <i>IXXX_Open</i> if an attempt is made to have more than 255 carriers or modules simultaneously open.
-9	E_BAD_PARAM	Returned by a function if it received an invalid parameter. This typically means the parameter was outside of the expected range or the function received a NULL pointer. Consult the individual function descriptions for other possible reasons for this error.
-10	E_INSUF_MEM	Returned by a function if there was insufficient memory to create a required data structure or carry out an operation.
-11	E_OCX_IN_USE	Returned by the ActiveX method <i>EnableIEvents</i> if the control is already configured for use by another IOS module
-12	E_DLL_LOAD	Returned by ActiveX methods if the IXXX DLL can not be loaded
-13	E_CONFIG_READ	Returned by <i>A86_ReadConfigReg</i> if an error occurred while reading data from the device's PCI configuration space.
-14	E_TIMEOUT	Returned by a function if it timed out before completing.
-15	E_CONFIG_SET	Returned by a Configuration function if the current settings used by this function do not represent a valid configuration
-16	E_CALIB	Indicates an error generating or using calibration data.
-17	E_BUFFER	Indicates an error occurred accessing a user defined data buffer
-22	E_FILE_OPEN	Returned if a file cannot be opened
-23	E_FILE_FORMAT	Returned if file contents are not in the expected format
-24	E_FILE_READ	Returned if a file cannot be read
-25	E_CONFIG_DONE	Returned by functions if the configuration of a re-configurable board is not complete

-26	E_EX_DESIGN	Some DLL functions for reconfigurable Acromag IOS modules will return this error if the module is not configured with Acromag example design.
-29	E_UNSUPPORTED	Returned if the hardware does not support the function.
-30	E_CHECKSUM	Returned if a checksum mismatch is detected
-31	E_HANDLER	The function requires that an interrupt handler be attached
-32	E_READY	The device is not ready to execute this command
-33	E_EXT_LIBRARY	An external library returned an error
-34	E_RESOURCE_IN_USE	The specified resource is already in use
-40	E_INVALID_CRHNDL	Returned if no IOS carrier is associated with the specified handle. Applies to most IOS carrier DLL functions
-41	E_EMPTY_SLOT	Returned by carrier and IOS module functions if the specified carrier slot does not contain a module. Note that this error is also returned if slot 'E' is specified. Only slots A – D are valid for the IOS Carrier
-42	E_SLOT_IN_USE	Returned by <i>IXXX_Open</i> if the IOS in the specified carrier slot is already open
-43	E_MODULE_MODEL	Returned by <i>IXXX_Open</i> if the IOS in the specified carrier slot is not a model supported by this DLL
-44	E_HANDSHAKE	Returned by serial communication functions if an expected handshake signal was not received

SEQUENCE OF OPERATIONS

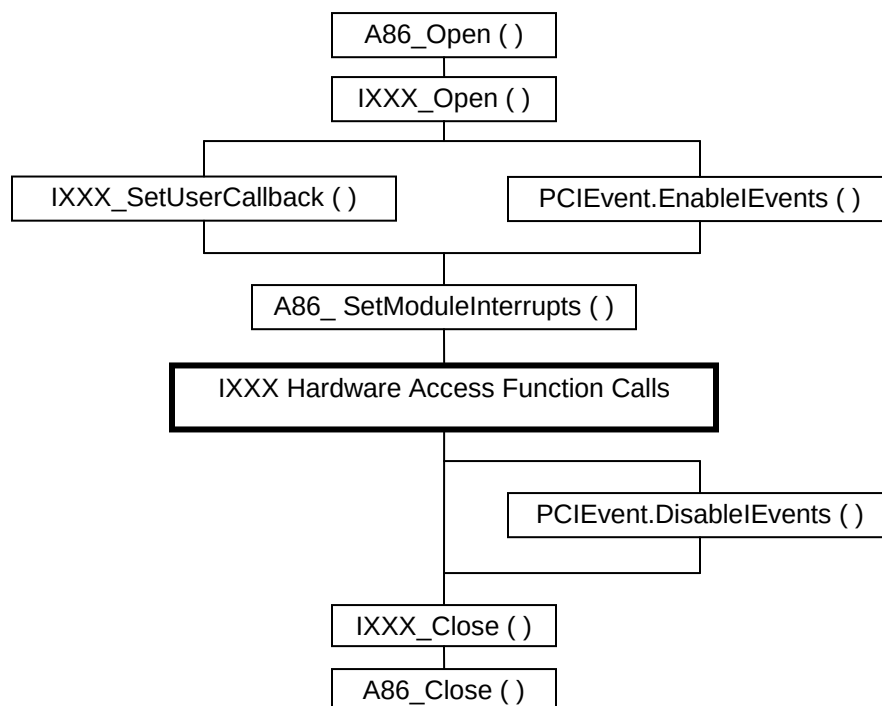
Although each IOS module has its own DLL with unique functions, all IOS modules are accessed in the following manner:

1. Open a connection to the carrier where the IOS module is installed by calling `A86_Open`. This function provides an integer "carrier handle" that is used in all subsequent carrier function calls.
2. Open connections to one or more IOS modules on the carrier by calling the corresponding `IXXX_Open` functions. `IXXX_Open` receives the carrier handle and carrier slot letter and provides an IOS module handle that is used in subsequent function calls to that module.
3. (Optional*) Setup a callback function or ActiveX event notification for each IOS that supports interrupts. (Callback functions and event notifications are mutually exclusive.)
4. (Optional) Enable the generation of IOS module interrupts by calling `A86_SetModuleInterrupts`.
5. Implement your application logic using the carrier and IOS module hardware access function calls.

Prior to terminating the application:

6. Call `DisableEvents` for each IOS module using ActiveX event notifications.
7. Close each IOS connection by calling `IXXX_Close`.
8. Close the carrier connection by calling `A86_Close`.

These steps are summarized in the diagram below



* A callback function or event handler is required when using an IOS-560 with interrupts. The I571 DLL does not provide a `SetUserCallback` function as callbacks can be configured through the Enhanced Mini-ACE Runtime Library.

Interrupts

IOS Win32 Driver Software provides two mechanisms for allowing your application to respond to interrupts generated by an IOS module: callback functions and ActiveX event notifications. These techniques are mutually exclusive. An application may implement one or the other for a particular module but not both. In most cases it is also acceptable for your application to not implement either option[†]. In general, the mechanism you use will be dictated by your choice of programming language. Callback functions are the preferred technique since they are synchronous, but they are not fully supported by all development environments.

Each IOS DLL that supports interrupts has its own predefined internal interrupt service routine. The specifics of each routine are outlined in the IOS module's corresponding Function Reference document. If you choose to implement a callback function or use ActiveX event notification, you have the option of overriding this routine. This is done by setting a *"Replace"* parameter when designating the callback function or during configuration of the ActiveX control. (See **Callback Functions** and **PCI Event ActiveX Control**)

When an interrupt occurs the following sequence of events takes place:

1. The kernel level driver disables the carrier's IOS Module Interrupt Enable bit and signals the carrier DLL's internal interrupt service routine (ISR).
2. The carrier ISR identifies the IOS module with pending interrupts and calls the device specific ISR in the corresponding IXXX DLL.
3. At this point four things can happen
 - If no callback or event notification was configured, the IXXX ISR simply processes the interrupt and returns a True value to the carrier ISR.
 - If a callback function or event notification was configured but should not override the internal ISR, the internal IXXX ISR processes the interrupt, invokes the callback or notifies the ActiveX control to fire an event and returns a True value to the carrier ISR.
 - If a callback function was configured to override the internal IXXX ISR, the ISR invokes the callback rather than process the interrupt and then returns a True value to the carrier ISR. It is the responsibility of the callback function to process the IOS interrupt.
 - If an ActiveX event notification was configured to override the internal IXXX ISR, the ISR notifies the ActiveX control to fire an event and returns False without processing the interrupt. It is the responsibility of the event handler to process the IOS interrupt and re-enable the carrier's IOS Module Interrupt Enable bit.
4. The carrier ISR identifies pending IOS interrupts in slot order (A – D). Steps 2 and 3 are repeated for each interrupting IOS module. If all IXXX ISRs have returned True, the carrier ISR re-enables the IOS Module Interrupt Enable bit.

Note that the carrier ISR will not re-enable the carrier's IOS Module Interrupt Enable bit if an interrupting IOS module is using an ActiveX event handler to override the IOS DLL's default ISR. This is because the event handler does not execute synchronously with the carrier ISR.

[†] A callback function or event handler is required when using an IOS-560 with interrupts. The I571 DLL does not provide a SetUserCallback function as callbacks can be configured through the Enhanced Mini-ACE Runtime Library.

CALLBACK FUNCTIONS

Callback functions are supported in C/C++ and Visual Basic .NET.

When using the callback mechanism your application defines a function that the IOS DLL will call from its internal interrupt service routine. The format of this function must exactly match that expected by the DLL. This format is hardware specific and is given in the **IXXX_SetUserCallback** topic in the IOS module's Function Reference document.

This format, however, will be some variation of the following:

```
C:          void (_stdcall *ISR)(short Handle, WORD Status)
VB.NET:    Sub ISR(ByVal Handle As Short, ByVal Status As Short)
```

The *Handle* argument identifies the IOS module that caused the interrupt. If the function is not overriding the internal ISR, the Status variable(s) will contain data allowing you to determine the cause of the interrupt (e.g. the value read from a status register by the internal ISR). If the function is overriding the internal ISR, the Status variable(s) will be zero since the internal ISR did not read any registers prior to invoking the callback function.

PCI EVENT ACTIVEX CONTROL

The PCI Event ActiveX control (**PCIEvent.ocx**) may be used for interrupt notification and processing in environments that do not support callback functions (LabVIEW) or where there are complications implementing thread safe code. In addition to the OCX file there is also a type library file (PCIEvent.tlb). Some development environments use this file to obtain information about the control's methods and events.

Note: Although the ActiveX control may also be used in Visual Basic .NET and windowed C/C++ applications, the callback approach is recommended since it is synchronous.

Instructions on adding the ActiveX control to an application and defining event handlers is deferred until the LabVIEW topic of the **Building Windows Applications** section. The general rules for using the control, however, are as follows:

1. The PCI Event control can only be associated with one IOS module at a time. Add one control to the application for each IOS module you wish to receive interrupt notifications from.
2. To associate the control with a board, call the ActiveX method *EnableIEvents* passing the IOS handle received from *IXXX_Open* and the IOS module's Model Code. Set the method's *Replace* parameter to indicate whether your event handler should override the DLL's internal ISR.
3. In many cases the PCI Event control can fire two types of events for the same interrupt condition. One type will pass argument(s), such as the value of a status register, which can be inspected to determine the interrupt source. The other type of event does not pass any arguments. The source of the interrupt can be determined from the event name.

For example, if interrupt conditions are sensed on input channels 0 and 6 of an IOS-408, the control will fire *Bit0*, *Bit6* and *PCIEvent1w(0x41)*.

Consult the DLL's Function Reference document to determine which events can be fired for your hardware. In general, applications will include just one of the handlers for a given interrupt condition. Which handlers you choose to implement will depend on the nature of your application.

4. If the event handler was configured to override the IOS DLL's internal ISR, the handler should call *A86_SetModuleInterrupts* to re-enable the carrier's IOS Module Interrupt Enable bit after servicing the interrupt.
5. Call *DisableIEvents* to disassociate the control from a board prior to calling *IXXX_Close* to close the module.

SYNCHRONIZATION

The DLL's interrupt service routine (ISR) executes on a different thread than that of your application. Within the DLL the ISR (which includes the call to any callback function) is delimited as a device critical section. *IXXX_StartIsrSynch* and *IXXX_EndIsrSynch* can be used to synchronize other application threads with the ISR thread. DLLs for Acromag IOS modules that do support interrupts also provide these functions to support the synchronization of hardware access by multiple threads within an application.

Bracketing a section of code between calls of *IXXX_StartIsrSynch* and *IXXX_EndIsrSynch* defines that code as a device critical section. Two threads within a single process cannot execute critical section code simultaneously. *IXXX_StartIsrSynch* should be called by your application before it attempts to access data or device memory that can be accessed by another thread. Remember to call *IXXX_EndIsrSynch* when finished accessing these shared resources.

Code in an ActiveX event handler function is not automatically defined as a critical section. If desired, *IXXX_StartIsrSynch* and *IXXX_EndIsrSynch* may be used to bracket this code and synchronize its execution with your application.

BASE ADDRESS POINTERS

Each IOS DLL provides a function that returns the base address of the user mode mapping of the IOS module's I/O space.

C and C++ programmers can cast the returned value to a byte pointer and access memory using normal pointer mechanisms. This method can be used to write additional functions that complement those provided through the DLL.

Example

```
/* Read IOS-408 Digital Input Channel Register B */

DWORD base_address;
volatile BYTE* pbase_addr;
WORD chan_val;

if (I408_GetBaseAddress(Handle, &base_address) == 0)
{
    pbase_addr = (BYTE*)base_address;
    chan_val = *(PWORD)(pbase_addr + 0x2);
}
```

Building Windows Applications

This section describes the basic steps to create applications that use the IOS Win32 dynamic link libraries and ActiveX control.

Steps are outlined for setting up projects in the following development environments:

- Microsoft Visual C++ 6 (VC6)
- Microsoft Visual C++.NET
- Visual Basic .NET
- LabVIEW 6i and 7

C/C++

Microsoft Visual C++ 6

The steps to set up a VC6 project to use IOS DLLs are as follows:

1. Open a new or existing Visual C++ project.
2. Add the path to the necessary header files (APC86xx.h, IXXX.h, IErrorCodes.h) to the project's **Preprocessor | Include** directories setting. This is located under **Project | Settings | C/C++**.
3. Add the path to the import libraries (APC86xx.lib, IXXX.lib) to the project.
To do this:
 - Open **Project | Settings | Link**
 - Select the **Input** category and modify the **Additional Library Path** field
 - Add the import library name to the **Object/library modules** field
4. Include the windows.h, APC86xx.h, IXXX.h and IErrorCodes.h header files at the beginning of your .c (C source code) or .cpp (C++ source code) files.
e.g.

```
#include <windows.h>
#include "APC86xx.h"
#include "I408.h"
#include "IErrorCodes.h"
```

Microsoft Visual C++ .NET

The steps to set up a VC.NET project to use IOS DLLs are as follows:

1. Open a new or existing Visual C++ project.
2. Add the path to the necessary header files (APC86xx.h, IXXX.h, IErrorCodes.h) to the project. To do this, open the project's property pages, open the **C/C++** folder, select the **General** property page and modify the **Additional Include Directories** property.
3. Add the path to the import libraries (APC86xx.lib, IXXX.lib) to the project.
To do this:
 - Open the project's property pages
 - Open the **Linker** folder, select the **General** property page and modify the **Additional Library Directories** property
 - Select the **Input** property page and add the import library to the **Additional Dependencies** property
4. Include the windows.h, APC86xx.h, IXXX.h and IErrorCodes.h header files at the beginning of your .c (C source code) or .cpp (C++ source code) files.
e.g.

```
#include <windows.h>
#include "APC86xx.h"
#include "I408.h"
#include "IErrorCodes.h"
```

VISUAL BASIC .NET

The steps to set up a Visual Basic .NET project to use IOS DLLs are as follows:

1. Open a new or existing project.
2. Add the files containing the DLL function prototypes (APC86xx.vb, IXXX.vb) and the error code constants (IErrorCodes.vb) to the project. To do this, select **Project | Add Existing Item** from the menu and select the desired files.

The following steps are only necessary if you will be implementing a callback function.

In Visual Basic .NET callback functions are implemented using delegates. A delegate is a class that can hold a reference to a method and is equivalent to a type-safe function pointer or a callback function.

1. Add a new module to the project that will contain the callback function. To do this, select **Project | Add Module** from the menu, select the **Module** template and select **Open**.
2. Add an ISR subroutine to the new module. The format of the routine is hardware specific and is given in the IXXX_SetUserCallback topic in the IOS module's Function Reference document.

This format, however, will be some variation of the following:

```
Sub ISR(ByVal IosHandle As Short, ByVal Status As Short)

End Sub
```

3. In the declares section of your form code declare a garbage collection handle:

```
Dim gch As GCHandle
```

(Note: If the editor indicates *GCHandle* is an undefined type, add `Imports System.Runtime.InteropServices` to the top of the source file.)

The DLL will store the delegate passed to it for later use. Since the DLL is unmanaged code, it is necessary to manually prevent garbage collection of the delegate until the DLL is through with it.

4. Include the following statements prior to the call to IXXX_SetUserCallback:

```
Dim dlg As IXXX_ISRDelegate    'defined in IXXX.vb
dlg = AddressOf ISR            'assign delegate to callback function
gch = GCHandle.Alloc(dlg)      'protect the delegate from garbage
                                'collection.
```

5. Now notify the DLL that it should invoke the callback

```
IXXX_SetUserCallback(IosHandle, dlg, fReplace )
```

6. At the end of your program, remember to free the garbage collection handle:

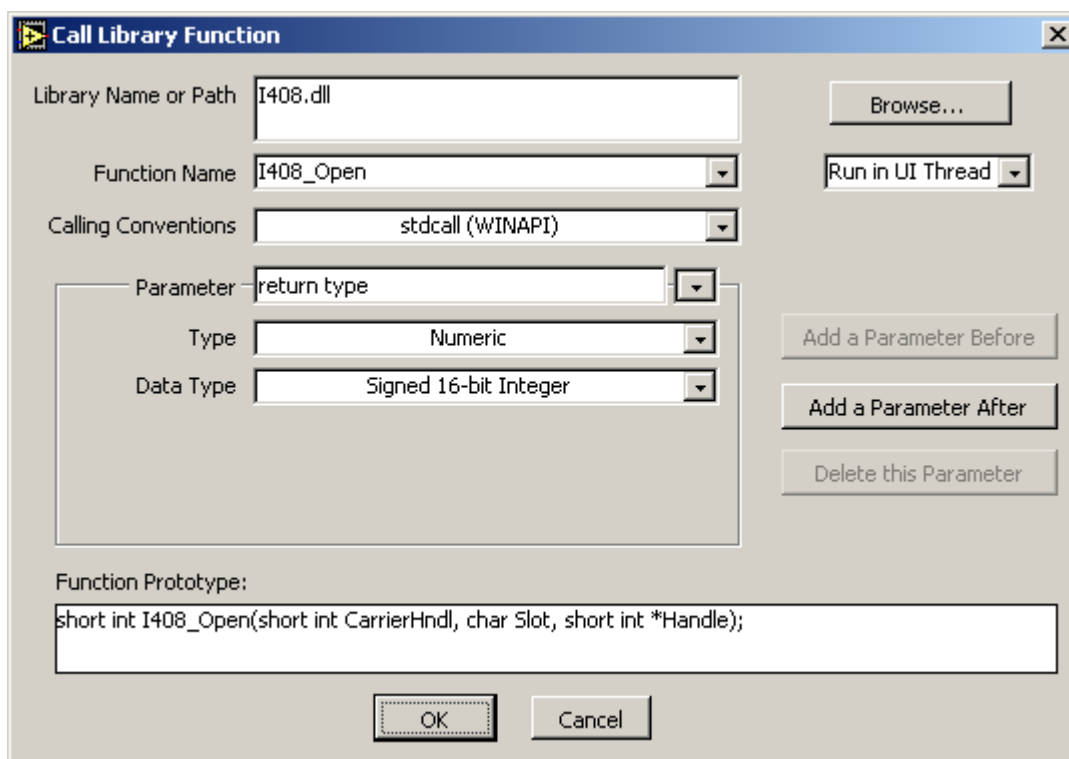
```
IXXX_Close(IosHandle)  
A86_Close(CarrierHandle)  
gch.Free()
```

LABVIEW

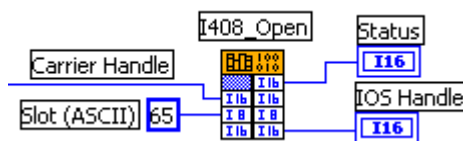
Calling a DLL function from LabVIEW 6i or 7

In LabVIEW DLL functions are called using Call Library Function nodes. The following steps outline how to add and configure one of these nodes.

1. Select **Call Library Function** from the **Functions | Advanced** subpalette and then click within the block diagram.
2. Right-click on the node and select **Configure**. This opens the Call Library Function configuration dialog.
3. Click the Browse button and locate the desired DLL. When finished, the name of the DLL will be displayed in the **Library Name or Path** field. Setting the Library Name in this manner automatically populates the **Function Name** Combobox.
4. Select the desired function from the **Function Name** Combobox.
5. Set the **Calling Conventions** to stdcall (WINAPI).
6. Leave the menu below the **Browse** button set to **Run in UI Thread** unless you plan on using the IsrSynch functions to make your DLL function calls thread safe.
7. Configure the return type as Numeric, Signed 16-bit Integer.
8. Add and configure additional parameters until the Function Prototype field matches the LabVIEW syntax listed for the function in the DLL's Function Reference guide. An example of a finished Call Library Function configuration dialog is shown below.



9. Wire inputs to the left side of the completed node and outputs to the right.



The following steps show how to use the PCI Event ActiveX control for event notifications in the VI.

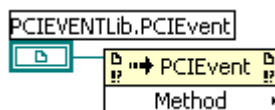
Adding the ActiveX control to a VI

1. LV6i: Select Container from the **Controls | ActiveX** subpalette and then click a point within the front panel to add the control to the panel.
LV7: Select ActiveX Container from the **All Controls | Containers** subpalette and then click a point within the front panel to add the control to the panel.
2. Right click within the container and select the PCIEvent control. After clicking **OK** the control will appear as a white box within the front panel. In addition, a node labeled "PCIEVENTLib.PCIEvent" will appear in the block diagram window.

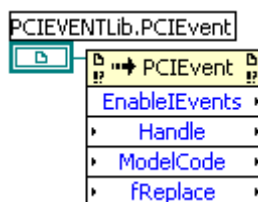
Invoking an ActiveX method

The methods, *EnableIEvents* and *DisableIEvents* are used to associate and disassociate the control from the handle (and hardware) used with the DLL functions. ActiveX methods are invoked as follows:

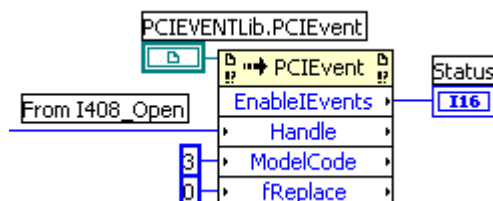
1. Select **Invoke Node** from the **Functions | Communication | ActiveX** subpalette and then click within the block diagram.
2. Wire the ActiveX control added previously to the **refnum** input on the node as shown below.



3. Right click on "Methods" to display a pop-up menu and select the desired method from the **Methods** sub-menu. The node will now show this method along with entries for its parameters.



4. Wire parameters (constants, type compatible controls and variables) to the left side of the node. Return values are wired to right of the method name.



Working with ActiveX Events (LabVIEW 6i)

The PCI Event ActiveX control fires events when an interrupt occurs in the control's corresponding IOS module. This section outlines how to monitor and respond to ActiveX events within the VI.

1. Select **Create ActiveX Event Queue.vi** from the **Functions | Communication | ActiveX | ActiveX Events** subpalette and then click within the block diagram.
2. Wire the previously added ActiveX control to the **refnum** input on the node.
3. Locate and right click on the **Event Name** terminal. Select **Create Constant** and type in the name of the event to monitor. This name can be obtained from the IOS module's Function Reference document. The name is case sensitive. The diagram should look as follows:

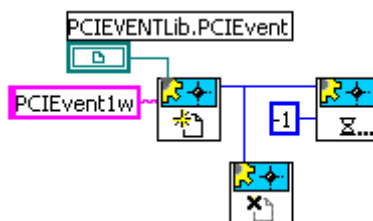


4. Select **Wait On ActiveX Event.vi** from the **ActiveX Events** subpalette and add the VI to the diagram. Wire the **Event Queue** terminals of the two VIs together. Locate the **ms timeout (-1)** terminal and add a constant to indicate the number of milliseconds the VI will wait for the event before timing out. A value of -1 means wait indefinitely.

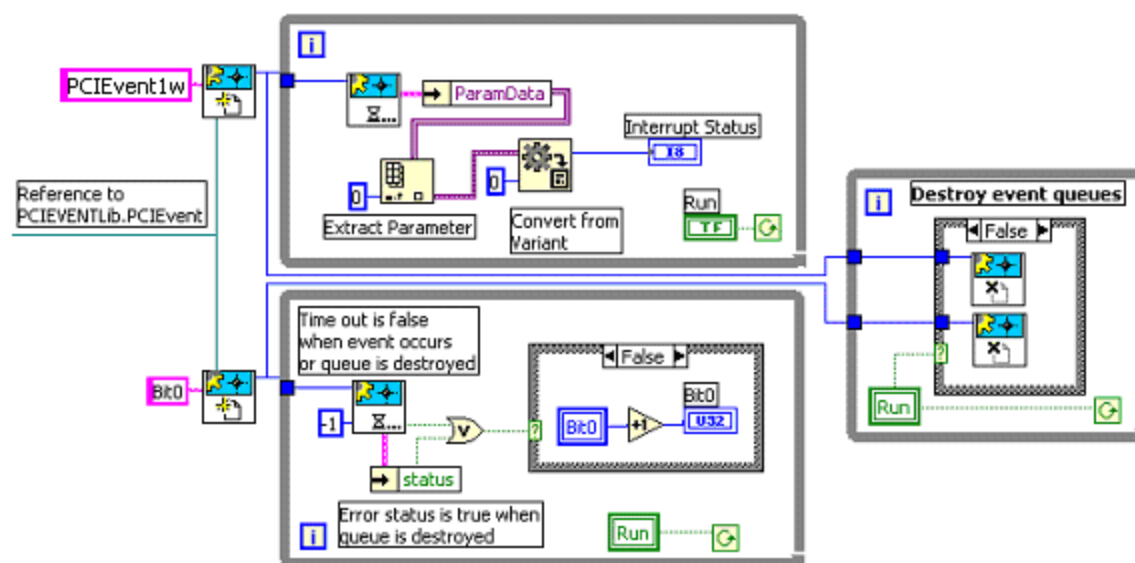
The output wiring from the VI will depend on if the selected event passes an argument or not. If the event passes arguments, wire the Event Data terminal, otherwise, wire the timed out terminal. (See complete example at the end of this section.)



5. Select **Destroy ActiveX Event Queue.vi** for the ActiveX Events subpalette and add the VI to the diagram. Wire the VI's **Event Queue** terminal to the **Event Queue (out)** terminal of the **Wait On ActiveX Event** VI. In the completed example, a control structure is used to determine when this VI executes. (See the complete example at the end of this section.)



6. Below is a complete block diagram showing the processing of events that do (PCIEvent1w) and do not (Bit0) pass arguments.



In the example above, the two Wait On ActiveX Event while loops execute as long as the Run button is activated. When the button is deactivated, both event queues are destroyed.

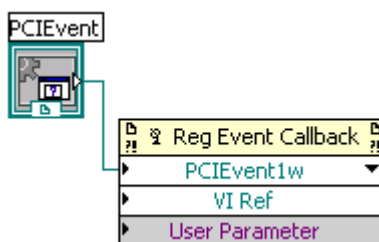
The upper **Wait On ActiveX Event** VI is used to monitor an event that returns a status variable. When an event occurs the VI returns the event data as a cluster. The parameter data and names are then unbundled as an array. Next, the parameter data is extracted from the array and converted from variant to integer data. Finally the status value is displayed in an indicator on the front panel.

The lower **Wait On ActiveX Event** VI is used to monitor Bit0 events. The VI's **timed out** terminal returns false when an event occurs or the event queue is destroyed. If the error status is also false, a Bit0 event indicator is incremented on the front panel. (The error status is set to true when the event queue is destroyed.)

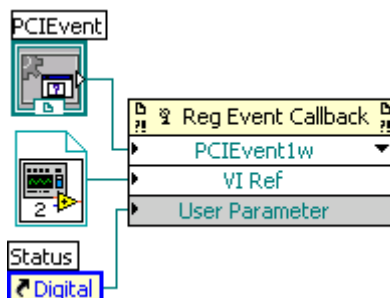
Working with ActiveX Events (LabVIEW 7)

The PCI Event ActiveX control fires events when an interrupt occurs in the control's corresponding IOS module. This section outlines how to register a VI to be called when a specific ActiveX event occurs.

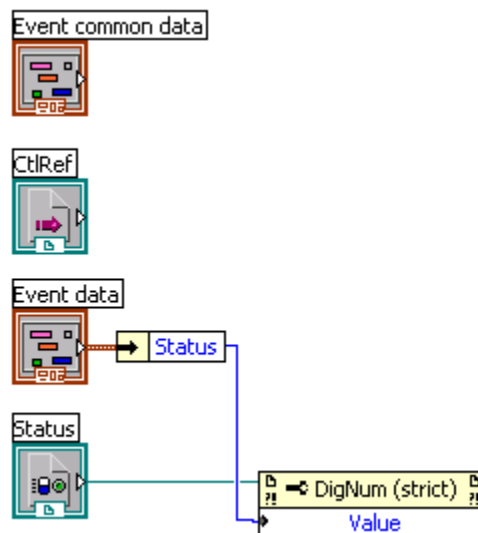
1. Select Register Event Callback from the **Functions | Communication | ActiveX** subpalette and then click within the block diagram.
2. Wire the previously added ActiveX control to the **event source ref** input on the node.
3. Click the down arrow next to the event source ref input and select the ActiveX event to be handled. The diagram should look as follows:



4. The User Parameter input can be used to transfer data between the callback VI and the main VI. For example, if a reference to an indicator on the front panel of the main VI is wired to this input, the callback VI can update the indicator each time the specified event occurs.
5. Right-click on the VI Ref portion of the node and select **Create Callback VI** from the context menu. At this point registration of the callback is complete. If necessary, the node can be resized to register callbacks for additional events.



6. The callback VI LabVIEW creates includes several nodes by default. Of primary interest are the nodes for the **User Parameter** and **Event Data**. The latter can be used to access arguments passed from the ActiveX control. Below is a complete block diagram for a callback VI that processes PCIEvent1w.



In the example above the value of the IOS-408's interrupt status register is passed from the ActiveX control to the Callback VI. The VI unbundles the status value from the Event data node and writes it to the Value property of the Status indicator (found on the front panel of the main VI). Note that if the callback VI handles an event that does not pass any arguments, the Event data node will not be present.

Deploying Applications to the I/O Server

The IOS Carrier driver, IOS DLLs and PCI Event Control are all preinstalled on the Windows Embedded Standard 2009 image for the I/O Server. This simplifies deployment of your custom application.

In the simplest case you will only need to copy an executable file to the Compact Flash drive. The system's PATH environment variable already includes the directory containing the IOS DLLs. This allows the system to locate the appropriate IOS DLLs when running your application.

Other custom applications may depend on additional files or system settings that are not present by default on the I/O Server. In this case the utilities listed in the **Installing Third-Party Applications** section can be used to help identify your application's requirements.

Debugging Applications on the I/O Server

Remote debugging of I/O Server applications is done over a TCP/IP connection between your development system and the I/O Server. How you go about remote debugging depends on a number of factors including: development environment used, native vs. managed code, user account permissions, network configurations, etc. Please refer to your development environment's documentation or online sources such as the Microsoft Developers Network for instructions on how to set up remote debugging.

Remote debugging has been tested and verified to work with Visual Studio 2005. Note that the Windows Embedded Standard image for the I/O Server has the Windows Firewall enabled by default. You will likely need to add exceptions to the firewall (or disable it) in order to debug your application.

Windows Embedded Standard Topics

Some users may have specialized requirements that are not met by the embedded image included with the I/O Server (IOS-7200-WIN or IOS-7400-WIN). In this case, users may wish to create their own Windows Embedded Standard 2009 image from scratch. A full discussion of Windows Embedded Standard is beyond the scope of this document. However, the following sections provide useful information should you wish to create your own custom image for the I/O Server.

TARGET ANALYZER

The Target Analyzer tool (Tap.exe) is used to probe the PCI bus, registry and BIOS of a target device. It creates a hardware inventory list (Devices.pmq) that is used to add required driver components to your configuration. Run TAP on the embedded image included with the I/O Server. This image already has all the necessary drivers installed so it will provide the most comprehensive information to TAP.

I/O CARRIER DRIVER

A device driver is required for the I/O Carrier card. The carrier driver can be included in the embedded configuration or installed manually after the system has booted for the first time. The default installation path for carrier driver files on the development system is:

```
C:\Acromag\IOSSW_API_WIN\redist\carrier_driver
```

Example steps for creating an I/O Carrier driver component and adding it to your configuration are provided below.

Creating a Device Driver Component for the I/O Carrier

1. Create a folder **C:\IOS_Carrier** on your development PC. This is where the component file (.sld) will be stored.
2. Create a sub folder **C:\IOS_Carrier\Repository** and copy Apc8620.inf and drvxdm.sys from **\Acromag\IOSSW_API_WIN\redist\carrier_driver** into it. This is the repository for the driver files.
3. Start Component Designer
4. Create a skeletal component from the INF file
 - From the **File** menu choose **Import**
 - Select File of Types "Setup Information Files (*.inf)"
 - Browse to **C:\IOS_Carrier\Repository**, select the Apc8620.inf file and click **Open**
 - An **INF Processing Options** dialog is displayed. Leave **Parsing Options** set to **Automatic** and click **OK**.
 - Click **Start** on the **Import File** dialog to import the .INF File.
 - After the import process has completed click **Close**.

5. Add a repository to the component
 - In the SLD browser, right-click the **Repositories** folder and then choose **Add Repository**.
 - In the **Repository Properties** dialog, change the default repository name to "**IOS Carrier Files**."
 - Click the **Browse** button next to the Source Path text box, browse to "**C:\IOS_Carrier\Repository**" and click **OK**.
 - In the SLD browser, select "Acromag Carrier" (under the Components folder) to display its properties.
 - Click the **Repositories** button on the right side of the Repository text box.
 - Select "IOS Carrier files" from the Select Repository dialog box, and then choose **OK**.
6. Use the File menu to save the component as **IOS_Carrier.sld** to the **C:\IOS_Carrier** folder you created previously.
7. Close Component Designer

Importing the I/O Carrier Component into the Component Database

1. Start Component Database Manager
2. From the Database tab, click **Import**
3. Browse to "**C:\IOS_Carrier\IOS_Carrier.sld**" and click **Import**
4. Once the component has been imported, close the Component Database Manager

Adding the I/O Carrier Component to the Configuration

1. Open your configuration in Target Designer
2. Locate the Acromag Carrier component in the Component Browser (**Hardware → Other**) and double click it to add it to the configuration

If you do not wish to create a component for the I/O carrier, the driver carrier can be installed after the system has booted from your custom image. The following steps assume your configuration includes the components necessary to run the Found New Hardware Wizard.

Installing the I/O Carrier driver on a live system

1. Make sure the carrier card is installed in the I/O Server and power up the system.
2. When the Found New Hardware Wizard appears select "Install from a list or a specific location (Advanced)" and click **Next**
3. Check the "Include this location in the search" box and browse to a folder that contains both the Apc8620.inf and drvxdm.sys files.
4. Click **Next** and the wizard will install the driver.

SYSTEM ON MODULE DRIVERS

Drivers are also required for devices (graphics, audio, LAN, etc.) present on the System On Module (SOM) board. Like the I/O Carrier driver, these drivers can be included in the embedded configuration or installed manually after the system has booted for the first time. Windows XP SOM drivers are provided in following subdirectories of the IOSSW_API_WIN installation:

IOS7200 (Geode): \redist\system_on_module\IOS7200

IOS7400 (Atom): \redist\system_on_module\IOS7400

Note:

These drivers are as provided by the third-party manufacturer of the SOM boards. Acromag currently does not have instructions for componentizing these drivers.

I/O CARRIER AND MODULE DLLS

The I/O Carrier DLL (APC86xx.dll) and the I/O module DLLs can be componentized and included in the embedded configuration or installed manually after the system has booted for the first time. These DLLs are dependent upon the Microsoft® C Runtime Library (msvcr71.dll). The default installation path for the DLL files on the development system is:

C:\Acromag\IOSSW_API_WIN\redist\DLLs

DLL Location Notes

To reduce the likelihood of "DLL Conflict" issues Microsoft recommends that DLLs be installed to the application directory with the program executable. This is the preferred location when running a single executable. However, if several applications will be simultaneously sharing a carrier or IOS DLL it is recommended that the DLL be placed in a common directory. This allows the shared DLL to properly track which boards are in use.

In order for the operating system to find a DLL, its location must be part of the Windows search order. The normal search order is as follows:

1. The directory from which the application loaded
2. The system directory (typically C:\WINDOWS\system32)
3. The 16-bit system directory (typically C:\WINDOWS\system)
4. The Windows directory (typically C:\WINDOWS\)
5. The current directory
6. The directories listed in the PATH environment variable

The easiest solution to sharing a DLL is to place it in the Windows or Windows system32 directory. However, many applications store DLLs in these directories so using these locations creates the most risk for DLL conflict issues.

In the Windows Embedded Standard 2009 image included with IOSSW-DEV-WIN, the common DLL directory (C:\Program Files\Acromag\IO Server\libraries) is appended to the PATH environment variable. This allows the appropriate DLL to be located when running each example project.

To manually modify the PATH variable

1. Select Start, Settings, Control Panel, and double-click System.
2. Select the Advanced tab and then click the Environment Variables button.

3. Locate "Path" in the User Variables or System Variables. The PATH is a series of one or more directories separated by semicolons.
4. Edit the variable by appending the path to the common DLL directory to the right of the existing value.
5. Click OK

PCI EVENT ACTIVEX CONTROL

The PCI Event ActiveX control (PCIEvent.ocx) can be componentized and included in the embedded configuration or installed manually after the system has booted for the first time.

The default installation path for the control on the development system is:

```
C:\Program Files\Common Files\Acromag\PCIEvent
```

This file is shared by IOSSW-API-WIN, IPSW-API-WIN and PCISW-API-WIN.

The ActiveX control is dependent upon the Microsoft® C Runtime Library (msvcr71.dll) and the MFC DLL Shared Library (mfc71.dll). By default these files are installed to

```
C:\Acromag\IOSSW_API_WIN\redist\DLLs
```

on the development system. Both DLLs need to be present on the I/O Server. See the **DLL Location Notes** topic above for information on where to place these files.

The ActiveX control needs to be registered on the I/O Server in order to function. ActiveX controls are registered using the Regsvr32 tool (regsvr32.exe) which can be found in the \WINDOWS\System32 folder. The simplest way to register the control is as follows:

1. Place a copy of regsvr32.exe into the same folder as PCIEvent.ocx.
2. Within Windows Explorer, use your mouse to drag PCIEvent.ocx on top of regsvr32.exe and release. Regsvr32.exe will display a dialog box indicating success or failure. Registration will fail if the previously mentioned Microsoft DLLs cannot be located by the system.
3. Delete regsvr32.exe

Regsvr32 can also be run from the command line:

```
regsvr32.exe PCIEvent.ocx
```